SMC-TR-97-16

AEROSPACE REPORT NO.
TR-96(8617)-1

Computer Systems Division Software System Metrics Approach Revision 1

September 1996

Prepared by

S. K. HOTING AND R. J. COSTELLO
Computer Systems Division
Engineering and Technology Group

Prepared for

SPACE AND MISSILE SYSTEMS CENTER
AIR FORCE MATERIEL COMMAND
2430 E. El Segundo Boulevard
Los Angeles Air Force Base, CA 90245

Engineering and Technology Group

THE AEROSPACE CORPORATION
El Segundo, California

This report was submitted by The Aerospace Corporation, El Segundo, CA 90245-4691, under Contract No. F04701-93-C-0094 with the Space and Missile Systems Center, P. O. Box 92960, Los Angeles, CA 90009-2960. It was reviewed and approved for The Aerospace Corporation by W. K. Clarkson, General Manager. The project officer is Colonel Charles E. Whited.

Charles E. Whited, Colonel, USAF
Project Officer

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

September 1996

3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE

Computer Systems Division Software System Metrics Approach, Revision 1

5. FUNDING NUMBERS

F04701-93-C-0094

6. AUTHOR(S)

S. K. Hoting and R. J. Costello

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Aerospace Corporation
2350 E. El Segundo Blvd.
El Segundo, CA 90245-4691

8. PERFORMING ORGANIZATION REPORT NUMBER

TR-96(8617)-1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Space and Missile Systems Center
Air Force Materiel Command
2430 E. El Segundo Boulevard
Los Angeles Air Force Base, CA 90245

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

SMC-TR-97-16

11. SUPPLEMENTARY NOTES

This Technical Report is a revision of Aerospace Technical Report TR-94(4904)-3, Space and Missile Systems Center Report SMC-TR-94-43.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Over the decades, the increasing use of software has enabled construction and deployment of ever more capable space systems for SMC. However, managing, controlling, and participating in software development for such software-intensive systems is a difficult and error-prone task that is exacerbated by the lack of meaningful data on key products and processes. Such data should provide visibility into the health and status of the evolving software system, assist in early identification of current and potential problem areas, and aid in predicting such software system characteristics as reliability, maintainability, cost, and schedule. Until recently, comprehensive data collection and analysis methods which treat software as an integral part of a larger system have not been available. This report introduces a software system metrics approach that has been developed for this purpose. The report discusses the utility of the approach, key concepts for applying metrics to software-intensive systems, and basic metrics planning guidelines. It also introduces a set of recommended metrics that cover both the system and the software throughout the life cycle. A complete description of the approach, associated contractual guidelines, and detailed descriptions of several metrics are contained in Aerospace TOR-96(8617)-1, *Metrics for Software-Intensive Mission Critical Computer Resource (MCCR) Systems.*

14. SUBJECT TERMS

Acquisition	Software	Software Metrics	System Life Cycle
Contract	Software Engineering	Software Process	System Metrics
Metrics	Software Lifecycle	Systems Engineering	Systems Engineering Process

15. NUMBER OF PAGES

13

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19 SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Contents

Tables

Preface to Revision 1

This Technical Report (TR) is a revised version of Aerospace Report No. TR-94(4904)-3. There are three main reasons this revision is being published. These are: (1) the increasing importance of metrics to our work in supporting the acquisition of large, software-intensive systems; (2) the continuing evolution of the metrics discipline; and (3) changes we have made to the recommended metrics sets to reflect this evolution and to reflect work we have done since the original version was published.

As space systems become more and more software intensive, it becomes increasingly important to be able to measure software system cost, schedule, development progress, and quality. At Aerospace, we are frequently asked to provide assistance both in developing and assessing contractors' metrics programs and in analyzing the metric data collected. We have few experts in this area, and to increase the effectiveness and timeliness with which we can respond to these requests, we need materials we can use as a foundation to guide all programs. Two products we have developed to this end include this TR and its newly published companion Technical Operating Report, Aerospace Report No. TOR-96(8617)-1, *Metrics for Software-Intensive Mission Critical Computer Resource (MCCR) Systems*. While the TR provides an overview of the recommended metrics approach, the TOR is the first extensive work that combines in-depth information on an approach, detailed metrics definitions, and planning and contractual guidance in using software and system metrics. The detailed definition for each metric addresses its purpose; the raw data to be collected; all calculations to be performed on the raw data; and collection, reporting, and interpretation/analysis procedures. Contractual guidance is included in the TOR in the form of tasking text and tailoring for software and systems engineering planning documents and reporting documentation.

Software and system metrics have also become more critical in the current acquisition reform environment. Acquisition reform dictates that we develop truly effective means to obtain insight into development effort health and status without the more labor-intensive oversight into the product that we have used in the past. In fact, current acquisition regulations encourage the use of software metrics.[1] However, little guidance is available on how to implement their use on major programs. By providing an approach and detailed information on the use of system and software metrics, this TR

[1] For example, DoD Regulation 5000.2-R, *Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs*, 15 March 1996, states that it is DoD policy to use software metrics "... to effect the necessary discipline of the software development process and assess the maturity of the software product."

and the above-referenced TOR can assist program offices in adopting acquisition reform without losing all insight into the systems for which they are responsible.

Revisions to this TR were also made necessary by the evolution of the metrics discipline and its application to software engineering. Still a very young field in comparison with other branches of engineering, software engineering has seen many methods and practices applied in an attempt to solve its problems, and it is not clear that any one way of building large software systems will be suitable in all cases. Hence, a viable discipline for measuring software products, processes, development resources, and development progress will find it necessary to adapt itself to the specific system and development effort it is measuring. Fortunately, there are many basic but critical measures that can be applied to most systems, with some modifications in the details of collection and reporting. Some examples include requirements volatility, defect density/inspection effectiveness, and problem report metrics. Both the TR and the TOR focus on these kinds of metrics. We have made some modifications to the recommended sets of metrics for systems and software since the original version was published (see Tables 1 through 5), and these are reflected herein.

1. Introduction

For today's large, software-intensive systems, the length of the development cycle and the number and complexity of technical and organizational interfaces create a great deal of uncertainty and risk. Additionally, for many of these systems, the Government's acquisition philosophy dictates that minimal standards and contractor controls be included in the contract, which results in the Government having little insight into the quality of the developing software-intensive product. It is, therefore, necessary to be able to objectively evaluate these systems during their development to determine whether or not they will meet requirements, schedule, and budget; to assist risk management; and to facilitate corrective and preventive action. Software system metrics can provide objective information necessary for technical and managerial insight into, control of, and improvement of the development effort.

Over the last few years, Computer Systems Division personnel have developed a metrics approach that has been designed for use during the development of large software-intensive systems. This approach includes an integrated set of system-level and software-level metrics recommended for collection by the development contractor(s) and detailed descriptions of each of these metrics. In creating this set of recommended metrics and their descriptions, the results of other current metrics technology efforts were incorporated, as appropriate. The metrics approach also includes suggested tailorings to selected contractual documentation to ensure that needed metrics information will be collected and reported to the Government. The metrics approach, recommended contractual documentation guidelines, and detailed descriptions of several of the recommended metrics are currently available in Aerospace Report No. TOR-96(8617)-1, *Metrics for Software-Intensive Mission Critical Computer Resource (MCCR) Systems*.

2. Metrics Within Large Systems: Three Key Concepts

Three basic concepts recommended for the development of large software-intensive systems include *seamlessness*, *consistency*, and *defined expectations*. These concepts apply to many aspects of development, such as supportability and reliability, as well as to metrics. The *seamlessness* concept recognizes that most of our software systems will be developed by a prime and several subcontractors. Seamlessness means that products developed by the various contractors should be uniform to the extent possible in order to increase the efficiency of communication among the contractors and between contractors and the Government, to reduce interface complexity, and to enhance maintainability and traceability. For similar software, uniform methods and types of tools and uniform training in these methods and tools are recommended. Thus, in accordance with the concept of seamlessness, all contractors should collect and report the same metrics information so that a uniform set of metrics information is reported to the program office.

The *consistency* concept recognizes that the total software process is an integral part of the overall systems engineering process and must be dealt with as such throughout the entire life cycle and across all systems engineering disciplines. The systems engineering process has a system-level component to the process, which then flows down to hardware- and software-level subprocesses. Consistency among these levels is necessary. For a large system, metrics should be collected on several levels: system, segment, and lower levels. Within the lower levels, there are hardware-specific and software-specific components. The software-level metrics program has been created to be consistent with and provide information to the higher level measurements. The higher level process will detail the methods by which lower level measures are incorporated into higher level measures. Software-level metrics are defined to be those that deal with software-only components; integrated hardware/software components are handled by higher level measurements.

The metrics approach includes effective, early communication of Government technical *expectations* to the contractor(s) before Engineering and Manufacturing Development (EMD) so that the contractor(s) can create appropriate plans to meet these expectations. It is recommended that this be done by: delivering Government expectations documents to the Demonstration/Validation (Dem/Val) or Pre-EMD contractors before they begin developing their EMD planning documentation; participating in Government-contractor Integrated Product Teams (IPTs); and providing feedback on early versions of developing planning documentation. One purpose of the TOR referenced in Section 1 is to provide a basis for conveying such metrics expectations to the contractor.

3. Metrics Planning: Basic Guidelines

3.1 Primitive and Aggregate Measures

The purpose of the metrics program is twofold: to gain visibility into the overall health and status of the evolving system and to identify, at the earliest possible point in the life cycle, specific problem areas or potential future problems. Both detailed and aggregate measures are necessary and need to be reported to the Government on a regular basis (often monthly). To assess overall health and status, cumulative measures should generally be used, whereas for the identification and resolution of problems, metrics should be reported at a detailed level. Detailed, or primitive, information should be reported (or made available) in electronic form for analysis and retention by the program office.

3.2 Metric Descriptions

Emphasis is placed on the need for careful definition and description of each metric and its report formats. Without specific definitions of precisely what is being measured, the measurement will have little meaning or use. It is, for example, insufficient to report source lines of code (SLOC), without discussing how that code is being counted. A definition that excludes data declarations and comments and counts only executable SLOC may easily result in a metric value that is half that resulting from a definition that includes data declarations and comments. Additionally, without relatively consistent descriptions of a given metric that is used on several different programs, it will not be possible to adequately evaluate the usefulness of reported metric data.

3.3 Metrics Collection/Reporting Tools

It is expected that whenever possible, the collection of metrics data will be automated and will use tools that have been integrated into the contractor's software engineering environment. In general, it is preferable to use commercial tools when they are available. However, for some metrics it may be necessary to use contractor-developed tools, either because there are no commercial tools that calculate the defined metric or because the contractor tool already supports some aspect of the existing development process and that aspect is being measured. For example, if the contractor has an existing automated problem report tracking tool, then accumulating metrics on problem reports may be done most efficiently by modifying the existing tool to collect the defined metric. On a given development effort, the same metrics tools should be used by all development contractors, and to the extent possible, all tools and methods should be compatible and integrated among all levels of the software system.

3.4 Contractor Metrics Plans

The contractor's process planning documentation (systems level and software level) should include a detailed and unambiguous definition of each metric and its report formats, or should reference Government-provided definitions and report formats that the contractor intends to use. The plans should also include descriptions of methods/tools used to collect, analyze, and report metric information, as well as a description of management's use of the collected metric information to assess and improve the software system product and the processes used to generate the product.

3.5 Metrics for Many Disciplines

For software, the metrics program is designed to share information with many software disciplines (e.g., risk management, Software Quality Assurance, testing, management, and problem reporting). The contractor's software planning document should discuss the various software organizations/activities that use metric data. The use of metric information to assess software risk, to assess and improve software processes, to manage the technical effort, and to identify error-prone software units should, for example, be explained.

4. Recommended Metrics

The activities of selecting and defining a set of metrics that effectively covers the software process can only reach closure in the context of the specific development processes to be used. However, it is possible to list a general set of software metrics which covers the main activities and phases of the software life cycle. This set can be tailored and specific metric definitions can be developed to suit a specific software life cycle and process.

Table 1 shows an example set of metrics that covers the software life cycle. Three categories of metrics have been identified: progress, resource, and product/process. A collection of metrics from each of these categories is usually required for comprehensive coverage. Progress metrics indicate an organization's adherence to schedule. Resource metrics indicate the amount of development, integration, test, and/or support resources and personnel available and the amount in use. Product/process metrics are used to measure attributes of the documentation (electronic and/or paper) and code and characteristics of the activities, methods, practices and transformations employed in developing the products. Product and process measurement activities tend to overlap, which is why they are combined into one category. For example, a high number of product defects can imply the existence of a problem in the process used to create the product. Also, a dearth of exposed defects can indicate the existence of a superior product or a deficient inspection process.

While it is necessary to have a software metric set that spans the software life cycle and is tailored to the process, this is not sufficient for a software effort that will be integrated into a larger system. Thus, we also recommend use of a set of progress and product/process metrics at the system level that is integrated and consistent with the software-level metric set, and these metrics are listed in Table 2. Summary descriptions of each type of metric listed in Tables 1 and 2 appear in Tables 3 through 5. Complete descriptions for several of the metrics are provided in Aerospace Report No. TOR-96(8617)-1, *Metrics for Software-Intensive Mission Critical Computer Resource (MCCR) Systems*.

Table 1. Recommended Metrics for the Software Measurement Program

PRODUCT AND PROCESS

Volatility
- –Requirements*
- –Design and Code
- –Build Content

Traceability
- –Between Requirements*
- –Between Requirements and Design
- –Between Requirements and Test

Problem Reports/Action Items/Issues (all products/processes)
- –Source of Error (Product)
- –Type of Error
- –Finding Activity
- –Severity of Error (Impact)
- –Criticality of Error (Priority)
- –Age
- –Status (Open-Unresolved/Open-Resolved/ Closed)
- –Reason for Closure

Defect Density/Inspection*
- –Requirements Defect Density
- –Design and Code Defect Density
- –Inspection Coverage and Effectiveness

Fault Density/Test*
- –Requirements Fault Density
- –Design and Code Fault Density
- –Test Coverage and Effectiveness

Interface Consistency
- –Requirements
- –Design and Code

Target Resource Utilization
- –CPU
- –RAM
- –DISK
- –I/O Channel

Size (for CSCI, CSC, and CSU)
- –Requirements (Specification Language Elements/Lines, Number of Requirements, etc.)
- –Design (Specification Language Elements/Lines)
- –Code (Source Language)*
  - --High-Order, Assembly, and Special Purpose Languages
  - --Operating System Command Languages
  - --Data Base Definition Languages
  - --User Interface Construction Languages
  - --Expert System Rules

PRODUCT AND PROCESS (continued)

Complexity (design and code)
- –Logic Structure*
- –Information Flow*
- –Database Structure
- –Coupling
- –Cohesion

PROGRESS

Completeness
- –Requirements Specification Completeness*
- –Design
  - --Design Document Completeness
  - --CSC/CSU Design Completeness
- –Code
  - --CSC/CSU Code Completeness
- –Test Document
  - --Test Plan Completeness
  - --Test Description Completeness
  - --Formal Qualification Test (FQT) Dry Run/ Rehearsal Completeness
- –Test Event
  - --CSU Unit Test Completeness
  - --CSC Integration Test Completeness
  - --CSCI Integration Test Completeness
  - --Build (Software Integration) Test Completeness
  - --FQT Completeness

Integrated Progress
- –Requirements
- –Design
- –Code
- –Test Document
- –Test Event

PROJECT RESOURCE

Staffing
- –Actual Vs. Planned Level/Turnover Rate
  - --Major Software Function
  - --CSCI
  - --Skill Level

Resource Utilization
- –Development/Integration/Test Resources
  - --CPU
  - --RAM
  - --Mass Storage (on-line/off-line)
  - --I/O Channel
  - --Workstation

*Definition exists in Aerospace Report No. TOR-96(8617)-1, *Metrics for Software-Intensive Mission Critical Computer Resource (MCCR) Systems.*

Table 2. Recommended Metrics for the System Measurement Program

PRODUCT AND PROCESS	PROGRESS
<u>Volatility</u> –Requirements* --System/Segment --Integrated Configuration Item (CI) --Hardware CI –Design --System/Segment --Integrated Configuration Item (CI) --Hardware CI <u>Traceability</u> –Between Requirements* --System to Segment --Segment to CI --Higher Level CI to Lower Level CI –Between Requirements and Design --System/Segment --Integrated CI --Hardware CI –Between Requirements and Test --System/Segment --Integrated CI --Hardware CI <u>Problem Reports/Action Items/Issues</u> (all products/processes) –Source of Error/Product –Type of Error –Finding Activity –Severity of Error (Impact) –Criticality of Error (Priority) –Age –Status (Open-Unresolved/Open-Resolved/Closed) –Reason for Closure <u>Defect Density/Inspection</u>* –Requirements Defect Density –Inspection Coverage and Effectiveness <u>Fault Density/Test</u>* –Requirements Fault Density –Test Coverage and Effectiveness <u>Interface Consistency</u> –System to External System Requirements –System to External System Design	<u>Completeness</u> –Requirements Specification Completeness* --System/Segment --Integrated CI --Hardware CI –Design --System/Segment Design Document Completeness --Design Completeness •System/Segment •Integrated CI •Hardware CI –Integration and Test --Test Document Completeness --Rehearsal Completeness --Test Event Completeness •System/Segment •Integrated CI •Hardware CI <u>Integrated Progress</u> –Requirements –Design –Implementation –Integration and Test

*Definition exists in Aerospace Report No. TOR-96(8617)-1, *Metrics for Software-Intensive Mission Critical Computer Resource (MCCR) Systems.*

Table 3: Product/Process Metrics

METRIC	SUMMARY DESCRIPTION: OVERVIEW AND PURPOSE
Volatility	Indicate changes in products/processes and reasons for change. Provide insight into system maturity and stability. Aid in predicting future changes to products/processes which are affected by current changes in products/processes. Essential in interpreting other metrics, e.g., progress, traceability, and completeness metrics. Recommended for requirements, design, code, and incremental build definitions.
Traceability	Indicate degree to which development organization maintains accountability for meeting requirements at each life-cycle stage via a comprehensive requirements allocation and mapping process. Measure relationships between requirements for a given product at a given level and: requirements at other specification levels; designs; code/databases; builds; and tests. Also measure relationships between designs for a given product and: code/databases; builds; and tests. Provide quantitative means for determining whether all required relationships/dependencies are addressed. Assist in exposing incompletely specified, insufficiently analyzed, overly specified, and complex areas of system. Essential in interpreting other metrics, e.g. completeness metrics.
Target Resource Utilization	Indicate planned and actual utilization of computer resources for target system. Provide timely feedback on whether software is being designed and developed to fit resources planned for its operational use. Assist in preventing adverse effects on cost, schedule, and quality due to inadequate system sizing. Recommended for CPU, primary memory, mass storage, I/O capacity, and other applicable resources.
Problem Report/ Action Item/Issue	Indicate quality of products, and processes used to create them; and effectiveness of engineering process in documenting and addressing problems, actions, and issues. Consist of counts of problem reports and action items characterized by source, product, problem type/category, age, severity, criticality, status, and primary reason for closure. Recommended for all products generated from requirements through testing and maintenance activities. Essential in interpreting other metrics.
Size	Indicate magnitude of development and maintenance effort. Used in assessing progress, estimating remaining cost and schedule, identifying technical problems, predicting maintenance cost and effort, generating historical data for future use, and quantifying the amount of reuse. Recommended for requirements, designs, and code. For code, size must include *all* code that the programmer writes in *any* language: compiled/assembled languages, operating system command languages, database definition languages, graphical user interface builders, and expert system shells. (SLOC is the recommended measurement for several of these languages.) Classified by: physical & logical statements, statement type, deliverable & non-deliverable statements, operational & support statements; and new, modified, & reused statements.
Complexity	Indicate structural characteristics of software system logic flow, information flow, and databases. Also indicate modularity of software. Useful in determining whether work has been completed satisfactorily, in planning for code development and test, in identifying technical problems, and in estimating development, test, and maintenance cost and effort. Several studies have shown that highly complex software is more likely to contain errors and is more difficult to maintain than less complex software.
Defect Density/ Inspection Effectiveness/ Inspection Coverage	Indicate density[2] of product defects that are detected *during an inspection or walkthrough*. Classified by type, criticality, and source. Provide early insight into quality, assist in cost/schedule estimation, indicate effectiveness of inspection/walkthrough process, and indicate the coverage provided in inspections (i.e., the amount of product covered in inspections). Recommended for requirements, designs, and code. Useful in predicting product/process volatility. Essential in interpreting other metrics, e.g., completeness, traceability, and volatility metrics.
Fault Density/ Test Effectiveness/ Test Coverage	Indicate density[2] of product faults that are detected *during test execution or post-test analysis*. Classified by type, criticality, and source. Assist in determining effectiveness of software process and quality of its products. Indicate test effectiveness and the coverage provided by tests (i.e., the amount of product covered in tests). Recommended for requirements, designs, and code. Useful in predicting product/process volatility. Essential in interpreting other metrics, e.g., completeness, traceability, test coverage, and volatility metrics. Provide data on product quality and compliance with requirements.
Interface Consistency	Indicate consistency and completeness of interface information at each level of specification.

[2] Density is the number of defects/faults found divided by the size of the product in which the defect/fault is detected.

Table 4: Project Resource Metrics

METRIC	SUMMARY DESCRIPTION: OVERVIEW AND PURPOSE
Staffing	Characterize number, discipline (e.g., design, coding, test, configuration management, quality assurance), skill level (discipline and years of education and experience), and area(s) of assignment (e.g., CSCIs) for development organization personnel. Indicate planned and unplanned changes in staffing level and assignments, which can be used to predict whether an effort is adequately staffed to preclude adverse effects on cost, schedule, and quality.
Development, Integration, and Test Resource Utilization	Indicate planned and actual utilization of computer resources for software development and support activities. Provide timely feedback on whether planned and available resources for each phase will adequately support the activities of that phase. Assist in preventing adverse effects on cost, schedule, and productivity due to resource shortages. Recommended for CPU, primary memory, mass storage, I/O capacity, workstations, and other applicable resources such as COTS software.

Table 5: Progress Metrics

METRIC	SUMMARY DESCRIPTION: OVERVIEW AND PURPOSE
Completeness	Indicate work accomplished versus work remaining in requirements and design specification, coding, inspection, unit test, integration and test, and system test. Assist in estimating cost and schedule remaining, in identifying technical problem areas, and in determining readiness to proceed to the next phase. Each class of completeness indicator (where a class focuses on a single product, e.g., requirements, design, code, or test) should be used in conjunction with the other measures for that class as indicated in the "Integrated Progress" metric description below.
Integrated Progress	Indicate overall progress in requirements, design, code, and test. Encompass measures of completeness, volatility, traceability, defect and fault density, problem reports/action items, and test coverage as appropriate for phase and product under consideration.

References

Costello, Rita J. and Sharon K. Hoting, et al. 1996. *Metrics for Software-Intensive Mission Critical Computer Resource (MCCR) Systems*. El Segundo, California: The Aerospace Corporation. Aerospace Technical Operating Report TOR-96(8617)-1.

Hoting, Sharon K. and Rita J. Costello. 1994. *Computer Systems Division Software System Metrics Approach*. El Segundo, California: The Aerospace Corporation. Aerospace Technical Report TR-94(4904)-3.

U.S. Department of Defense. 1996. *Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs*. Washington, D.C.: Office of the Secretary of Defense. Department of Defense Regulation 5000.2-R.

THE AEROSPACE
CORPORATION
2350 E. El Segundo Boulevard
El Segundo, California 90245-4691
U.S.A.